VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Procesor JDD jazyka SQL pro databázový systém RadegastDB

An SQL DDL Procesor for RadegastDB Database System

2017 Tomáš Grutman

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Grutman**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Procesor JDD jazyka SQL pro databázový systém RadegastDB
An SQL DDL Procesor for RadegastDB Database System

Jazyk vypracování: čeština

Zásady pro vypracování:

Databázový systém RadegastDB vyvíjený na Katedře informatiky poskytuje různé datové struktury pro řešení odlišných problémů uložení dat. V rámci této diplomové práce bude student implementovat procesor JDD jazyka SQL pro tento databázový systém.

1. Nastudujte fyzický model RadegastDB a jazyky pro definici dat.
2. Naimplementujte základní konstrukce JDD jazyka SQL.
3. Poveďte experimenty a výsledky porovnejte s existujícími implementacemi jiných databázových systémů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Michal Krátký, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

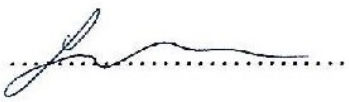
doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. června 2017



Rád bych na tomto místě poděkoval svému vedoucímu práce, doc. Ing. Michalovi Krátkému, Ph.D za cenné rady, které mi poskytl a za pevné nervy při konzultacích.

Abstrakt

Bakalářská práce se zabývá vrstvou nad embedded databázovým systémem RadegastDB, která nám umožní přeložit pomocí překladače jazyka definice dat SQL příkaz uživatele a vytvoří vše potřebné pro vytvoření požadované datové struktury. Následně tuto datovou strukturu pomocí RadegastDB vytvoří a nakonec uloží vše potřebné k tomu, abychom mohli kdykoliv tuto strukturu dotazovat nebo plnit daty. K vyřešení tohoto problému jsem využil materiály o již existujících částech databázových systémů a konzultace s vedoucím práce. Mé řešení umožňuje vytvoření zadaných datových struktur, které mnou vytvořený generátor dat naplní náhodnými daty a následně tyto struktury dotazujeme. Celý tento proces je měřen a vypsán pro zjištění doby trvání a efektivity.

Klíčová slova: RadegastDB, embedded databázový systém, jazyk definice dat, datová struktura, překladač, B-strom, R-strom, shlukovaná tabulka, systémový katalog, SQL, vzory, halda, space descriptor

Abstract

This Bachelor thesis deals with a layer above embedded database system RadegastDB, which allow us to translate an SQL input of a user using translator of data definition language and creates everything what we need for construction of our required data structure. Then, using system RadegastDB, it creates and saves everything what we need for querying or filling the data structure with data at any time. To solve this problem, I have used materials about existing parts of database systems and consultations with my supervisor. My solution allow us to create assigned data structures, which are filled with random data by my data generator and after that we are qurying these structures. This whole procces is measured and output is returned - to determining the efficiency and operation processing time.

Key Words: RadegastDB, embedded database system, data definition language, data structure, translator, B-tree, R-tree, clustered table, system catalog, SQL, patterns, heap, space descriptor

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **9**

Seznam tabulek **10**

1 Úvod do databázových systémů **11**
1.1 Cíl práce a struktura práce 11

2 Klasifikace databázových systémů **13**
2.1 Klasifikace podle modelu dat 13
2.2 Klasifikace podle typu distribuce 14
2.3 Další klasifikace 16

3 Dotazovací jazyky **17**
3.1 Jazyk SQL 17
3.2 Jazyk pro definici dat 17
3.3 Jazyk pro manipulaci dat 18
3.4 Použité datové struktury 19

4 Systémový katalog databáze **20**
4.1 Tabulky systémového katalogu existujících databázových systémů 20

5 Embedded databázový systém RadegastDB **22**
5.1 Využité datové struktury 22
5.2 Datové typy a space descriptor systému RadegastDB 22
5.3 Cache Buffer 23

6 Analýza implementace nástavby JDD **24**
6.1 Překladač JDD 24
6.2 Zápis do systémového katalogu 24

7 Fyzická implementace databázových systémů **25**
7.1 Fyzický návrh 25
7.2 Použité vzory fyzického návrhu 25
7.3 Základní plány vykonání dotazu 26
7.4 Příklad vytvoření struktury pomocí systému RadegastDB 26

8 Implementace **28**
8.1 Postup vytváření jednotlivých vzorů 28
8.2 Třídní diagram 29
8.3 Popis důležitých tříd a jejich metod 30

9 Testování jednotlivých vzorů **33**
9.1 Úvod testování jednotlivých vzorů 33
9.2 Vzor CREATE TABLE (B-tree) 34
9.3 Vzor CREATE TABLE (R-tree) 35
9.4 Vzor CREATE TABLE (shlukovaná tabulka) 35
9.5 Vzor CREATE INDEX 36
9.6 Výsledky testů 38
9.7 Porovnání s již existujícími databázovými systémy 38
9.8 Podporované datové typy embedded databázového systému RadegastDB 39

10 Závěr **40**

Literatura **41**

Přílohy **44**

A Příloha na CD **45**

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
JDD	–	Jazyk pro definici dat
DS	–	Datová struktura
JMD	–	Jazyky pro manipulaci dat
JDD	–	Jazyky pro definici dat
MBB	–	Minimal bounding box

Seznam obrázků

1 Distribuované databázový systém [6] 14
2 Ukázka cache bufferu 23
3 Sekvenční diagram: Rozpoznání typu příkazu 28
4 Třídní diagram 30
5 Vzorový výpis vzorů 1, 2 a 3 34
6 Vzorový výpis vzoru 4 37

Seznam tabulek

1 Relace *oddělení* 13
2 Relace *zaměstnanec* 13
3 Porovnání pohledů systémového katalogu 21
4 Použité datové struktury RadegastDB 22
5 Výsledky testování jednotlivých vzorů 38
6 Podrobný výpis času trvání jednotlivých úseků 38
7 Srovnání embedded databázových systémů 39
8 Obsah CD 45
9 Struktura projektu po extrakci knihovny boost 45

1 Úvod do databázových systémů

Kniha Databázové systémy [8] říká: Před vznikem databázové technologie v 70.–80. letech 20. století lidé chtěli ukládat velké množství dat, pro jejich pozdější dotazování a případně analýzy. Příkladem může být zpracování výsledků voleb, atd. S rozvojem výpočetní techniky začaly vznikat programy, které se různým způsobem snažily řešit uložení dat. Tyto programy pracovaly podle principu agendového zpracovávání dat. Bylo tedy nemožné dotazovat se uložené databáze bez znalosti datových struktur uložiště. Dalším problémem byl neexistence dotazovacího jazyka vyšší úrovně. Později začaly vznikat první datové modely specifikující, jakým způsobem budou data modelována a dotazována, stále však neexistoval nějaký dotazovací jazyk vyšší úrovně.

Vývoj databázových aplikací umožnil sjednotit pohled na pojmy, nástroje a metody, tedy databázovou technologii. Základ je databáze což je množinou vzájemně propojených dat, které jsou uloženy v daném databázovém systému. Aplikace, která umožňuje databázi definovat, konstruovat a manipulovat s databází se nazývá systém řízení báze dat (SŘBD, angl. DBMS). SŘBD spolu s databází tvoří databázový systém (DBS).

Databázový systém (neboli Systém řízení báze dat) je programový systém na efektivní ukládání, modifikaci a výběr velkého množství perzistentních dat. Databázové systémy se využívají v mnoha oborech (např. hry, online aplikace,firemní systémy, bankovní systémy, atd.) a měly by být: masivní (schopnost pracovat s neporovnatelně větším objemem dat než se vleze do hlavní paměti počítače), perzistentní, bezpečné, více-uživatelské, pohodlné, efektivní, spolehlivé.

Nejznámější databázové systémy: Oracle Database [23], Microsoft SQL Server [25], MySQL [24], IBM DB2 [26], Microsoft Access [27].

1.1 Cíl práce a struktura práce

Cílem této bakalářské práce bylo implementovat a otestovat vrstvu pro definici dat nad embedded databázovým systémem RadegastDB [21], který je vyvíjen na Katedře informatiky. Tato vrstva bude schopna přeložit JDD příkaz a zpracovat ho tak, aby byly data správně uložena v RadegastDB [22]. Následně jsme pak schopni znovu přistupovat k vytvořeným DS, vkládat do nich data, vytvářet nové DS a ukládat záznamy o vytvořených datových strukturách do systémového katalogu.

V této práci jsou popsány existující klasifikace databázových systémů a jejich příklady (kapitola 2). Nejsou zde uvedeny všechny, ty které jsou již zastaralé a nepoužívají se, jsem neuváděl. V této kapitole je i popsán hlavní objekt mé práce a to embedded databázové systémy. Je zde řečeno, co to vlastně je embedded databázový systém a zmiňuji se o jeho historii a o již existujících embedded databázových systémech. Dále pak je v kapitole 3 uveden popis dotazovacích jazyků a jejich součástí: jazyku pro definici dat, jazyku pro řízení dat, popis jazyka SQL i s jeho historií a popis používaných datových struktur. Následně se v kapitole 4 zmiňuji o systémovém katalogu, co to vlastně je a jaké by měl mít vlastnosti. V podkapitole se zmiňuji o specifikaci

systémového katalogu existujícího systému. V další kapitole 5 se nachází popis embedded databázového systému RadegastDB. Rozebírám tam jen struktury a datové typy, které využívám, jinak je RadegastDB mnohem rozsáhlejší. Popisuji zde i pojem cache buffer. Kapitola 6 popisuje můj překladač JDD a způsob ukládání dat do systémového katalogu aplikace. V kapitole 7 se nachází fyzický návrh mé aplikace, popis funkcionality použitých vzorů, popis plánu vykonání dotazu pro Oracle. Dále je tu také příklad vytváření datové struktury typu B-strom pomocí systému RadegastDB. Předposlední kapitola 8 obsahuje postup vytváření jednotlivých vzorů, třídní diagram aplikace a popis důležitých tříd a jejich metod. Poslední kapitola 9 zobrazuje výsledky testovaní. V podkapitole 9.7 pak porovnávám podporované datové struktury, které se objevují v mé práci a podporované datové struktury vybraných embedded systémů popsaných v kapitole 2.2.

2 Klasifikace databázových systémů

Existuje několik typů klasifikací [6]. Zde budou uvedeny ty nejznámější a jejich příklady.

2.1 Klasifikace podle modelu dat

Datový model [14] představuje zobrazení konceptuálního modelu do modelu, který je blíže fyzické implementaci databázového systému. Tento model nám určuje, jakým způsobem budou data modelována. Ze způsobem modelování úzce souvisí i jejich dotazování.

Relační databáze - Toto je momentálně asi nejpopulárnější používaný model. Je založený na SQL. Je tabulkově orientovaný, což znamená, že data jsou uložena v tabulkách, které obsahují klíč. Tento klíč slouží k identifikaci jednotlivých řádků. Mezi tabulkami jsou takzvané vazby, které nám pomáhají označit řádek nebo záznam a sloupce odkazující se na atributy tabulky. Několik příkladů: MySQL (Oracle, open source), Oracle database (Oracle), Microsoft SQL server (Microsoft) a DB2 (IBM).

Tabulka 1: Relace *oddělení*

id	**ulice**
1	Provaznická
2	Místecká
3	Zahrádky

Tabulka 2: Relace *zaměstnanec*

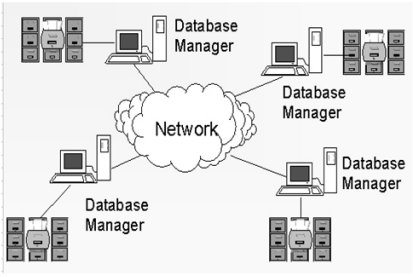
rodné číslo	**jméno**	**příjmení**	**funkce**	**id odd.**
7403285963	Petr	Novák	IT	3
1234567813	Jan	Přibyl	Bezpečnost	2
5921885184	Tomáš	Nový	Asistent	1
6541568545	Jana	Mladá	Vedoucí	1

V tabulkách 1 a 2 vidíme, jak jsou na sobě dvě tabulky pomocí vazby závislé. Oddělení má primární klíč id, nicméně slouží i jako cizí klíč v tabulce zaměstnanec, díky kterému je tato relace vytvořena.

Objektový a objektově relační datový model [6] - Přidává funkcionalitu databáze do objektově orientovaných jazyků jako uživatelské typy, dědičnost, metody tříd a rozlišení pojmu instance a ukazatel na instanci [8]. Je zapotřebí méně kódu a údržba zdrojového kódu je občas lehčí. Tyto výhody však 90% aplikací nemusí nutně využívat. Proto je častěji výhodnější a elegantnější využívat relační model.

2.2 Klasifikace podle typu distribuce

Distribuované databázové systémy [15] - Distribuovaný databázový systém je množina uzlů počítačové sítě, vzájemně propojených v síti přičemž každý z uzlů je samostatný databázový systém. Tyto uzly navzájem spolupracují tak, že z každého uzlu je možné zpřístupnit údaje uložené na jiném uzlu přesně tak, jak byly uloženy na vlastním uzlu. Samotná databáze tohoto databázového systému je množina navzájem propojených databází, které jsou umístěny na různých uzlech tak , že uživatel s nimi manipuluje, jako by byly uloženy v centralizované databázi. O řízení této databáze se stará distribuovaný databázový systém.



Obrázek 1: Distribuované databázový systém [6]

Klient-server databázový systém [6] - Tento systém má dvě logické komponenty. Klienta a server. Klienti jsou převážně osobní počítače nebo pracovní stanice zatímco servery jsou větší pracovní stanice nebo mainframe počítače. Aplikace a nástroje databázového serveru běží na klientech a samotný samotný databázový systém běží na serveru. Server a klient jsou propojeni přes síť. Komunikace probíhá tak, že aplikace nebo nástroj klienta pošle požadavek na službu severu. Server tento požadavek vyhodnotí a pošle klientovi odpověď.

Embedded databázové systémy - Tento systém je přesný opak klient-server databázových systémů. Embedded databázový systém [7] je databázový systém, který je přímo integrovaný s aplikací, díky kterému můžeme k datům přistupovat, ale databázový systém je "skrytý" pro koncové uživatele databázového systému a potřebuje jen velmi malou nebo žádnou údržbu. Jinými slovy embedded databázový systém nepotřebuje server a je vestavěný do aplikace.

Jedná se o širokou technologickou kategorii která zahrnuje:

- Databázové systémy s různými aplikačními programovými rozhraními
- Databázové modely (relační atd...)
- Cílový trh (podle požadavku zákazníka na funkcionalitu systému)

Tyto databáze jsou určeny pro práci v režimu single-user (jediný uživatel). Výhody užívání embedded databázových systémů jsou následující: jednoduché správa, malá velikost (ve

srovnání s velkými databázovými systémy např. SQL serverem), odladěnost, malá údržba kódu, rychlost.

Dříve se tyto databázové systémy [5] používaly téměř výhradně. Operační systémy MS-DOS a CP-M byly jednouživatelské a počítačové sítě téměř neexistovaly. Proto omezení těchto databází na single user přístup nikomu nevadilo. S nástupem LAN sítí se přesunuly databázové soubory z lokálních disků na síťové, implementovaly se jednoduché zamykací mechanismy a tyto databáze se ještě nějakou dobu hojně používaly. Pak se zjistilo, že jejich provozování přes síť není ideální a na LAN sítích se přešlo na architekturu klient-server. Důvodů pro to bylo několik. Zaprvé bylo nutno přenášet po síti značné množství dat, protože se zpracovávala na straně klienta a ne tam, kde byla uložena. Zadruhé tu byl problém s přístupovými právy (která prakticky nebyla implementována) a za třetí tu byl problém spolehlivosti – výpadek stanice mohl ohrozit konzistenci dat na serveru. Embedded databázové systémy stále převyšují počtem instalací klient-server databázové systémy.

Mezi hlavní embedded databázové systémy patří:

Berkeley DB od Oracle Corporation Berkeley DB [28] je embedded databázový systém od firmy Oracle Corporation, který koupil od Sleepycat Software. Berkeley DB je rychlá, open-source embedded databáze a je používána v několika známých open-source produktech, jako je například operační systémy Linux a BSD, Apache Web server [46], OpenLDAP directory [47] a OpenOffice [48].

Firebird Embedded - Firebird Embedded [29] je relační databázový systém. Jedná se o open source databázový systém, který je v souladu s ACID [18], podporuje triggery a uložené procedury a je dostupný pro Linux, OSX a Windows.

SQLite - SQLite [30] je softwarová knihovna, která implementuje samostatný, embedded, zero-configuration, transakční SQL databázový systém. Jedna se o nejrozšířenější SQL databázový systém na světě. Zdrojový kód, především v C pro SQLite, je na veřejné doméně volně ke stažení. Ve staženém balíčku se nachází nativní C knihovna a jednoduchý klientský příkazový řádek pro SQLite databázi. SQLite se využívá v několika operačních systémech: Android, FreeBSD, iOS, OS X a Windows.

Další příklady embeddeddatabázových systémů jsou: CSQL od csqlcache.com [31], eXtremeDB od McObject [32], HSQLDB od HSQLDB.ORG [33], Informix Dynamic Server (IDS) od IBM [34], InfinityDB od Boiler Bay Inc [35], InnoDB od Oracle Corporation [36], InterBase od Embarcadero Technologies [37], RDM Embedded od Raima [38], SQL Server Compact od Microsoft [39].

2.3 Další klasifikace

Dále se databázové systémy dělí podle počtu uživatelů, cíle využití (systémy bežící na mobilech, systémy na ukládání multimedií atd.) a samozřejmě dělení podle ceny jednotlivých systémů.

3 Dotazovací jazyky

Dotazovací jazyky [11] představují standardizovaný nástroj pro komunikaci mezi aplikacemi a databázovými systémy, tudíž nám umožňuje ovládat databázi pomocí příkazů. Máme dva druhy jazyků a to jazyk pro definici dat a jazyk pro řízení dat. Existuje mnoho dotazovacích jazyků, ale dnes je nejpoužívanějším dotazovacím jazykem SQL. Další známé dotazovací jazyky jsou XQUERY [40] pro XML, LINQ [41], Datalog [42], OQL [43], QUEL [44] a další.

3.1 Jazyk SQL

SQL [9] (anglicky Structured Query Language) je standardizovaný strukturovaný dotazovací jazyk, který je používán pro práci s daty v relačních databázích. SQL je nástupcem jazyka SEQUEL (anglicky Structured English Query Language).

3.1.1 Historie SQL

V 70. letech 20. století probíhal ve firmě IBM výzkum relačních databází. Bylo nutné vytvořit sadu příkazů pro ovládání těchto databází. Vznikl tak jazyk SEQUEL (Structured English Query Language). Cílem bylo vytvořit jazyk, ve kterém by se příkazy tvořily syntakticky co nejblíže přirozenému jazyku (angličtině).

K vývoji jazyka se přidaly další firmy. V r. 1979 uvedla na trh firma Relational Software, Inc. (dnešní Oracle Corporation) svoji relační databázovou platformu Oracle Database. IBM uvedla v roce 1981 nový systém SQL/DS a v roce 1983 systém DB2. Dalšími systémy byly např. Progres, Informix a SyBase. Ve všech těchto systémech se používala varianta jazyka SEQUEL, který byl přejmenován na SQL.

Zatím nejnovějším standardem je SQL:2016 [45] což je osmá revize standardu ISO (1987) a ANSI (1986) pro SQL.

Bohužel různé databázové systémy se více či méně od standardu odlišují. Při implementaci jádra mnoha systémů neobsahoval SQL standard všechny potřebné konstrukce a pro implementaci některých funkcí se výrobci museli rozhodnout pro vlastní řešení. Když byla následně daná konstrukce standardizována v systému již byla většinou ponechána původní verze. Rozdíly najdeme zejména mezi datovými typy, jmény a parametry funkcí a některými konstrukcemi z jazyka pro definici dat. Důsledkem je špatná přenositelnost databáze mezi jednotlivými databázovými systémy a nutnost zvýšené pozornosti k syntaxi [8].

3.2 Jazyk pro definici dat

Koncept JDD [10] a jeho název byl poprvé představen v souvislosti s Codasyl [16] databázovým modelem. Později se začal používat v SQL pro vytváření tabulek a integritní omezení, které zajišťují integritu databáze. Tyto integritní omezení (například záznam musí být v určitém

nadefinovaném rozsahu) zajišťují integritu databáze a za splnění těchto integritních omezení zodpovídá databázový systém. Jazyk pro definici dat je syntaxí podobný programovacím jazykům a používá se pro definici DS v databázových schématech.

První z příkazů JDD je příkaz CREATE. Tento příkaz se používá při vytvoření nové databáze, tabulky, indexu, pohledu nebo uložené procedury.

- CREATE TABLE pro vytvoření tabulky
- CREATE VIEW pro vytvoření pohledu
- CREATE INDEX pro vytvoření indexu
- CREATE PROCEDURE pro vytvoření uložené procedury

Další z příkazů je ALTER. Ten slouží k provedení změn v již už existující struktuře.

- ALTER TABLE změna definice tabulky
- ALTER VIEW změna definice pohledu
- ALTER INDEX změna definice indexu
- ALTER PROCEDURE změna definice procedury

Poslední z příkazů je DROP, který slouží pro mazání vytvořených struktur z databáze.

- DROP TABLE pro odstranění tabulky (nejen dat, ale i struktury tabulky a uvolnění paměťového prostoru, který byl pro tabulku vyhrazen)
- DROP VIEW pro odstranění pohledu
- DROP INDEX pro odstranění indexu
- DROP PROCEDURE pro odstranění uložené procedury – pokus o volání procedury od této chvíle skončí chybou

Všechny tyto příkazy se mohou lišit v závislosti na typu databáze, zde jsou ukázány běžné SQL příkazy.

3.3 Jazyk pro manipulaci dat

Jazyk pro manipulaci dat [12] je syntaxí podobný programovacím jazykům a používá se pro výběr, vkládání, mazání a aktualizování dat v databázi.

- SELECT ... FROM ... WHERE ...
- INSERT ... INTO ... VALUES ...
- UPDATE ... SET ... WHERE...
- DELETE FROM ... WHERE ...

3.4 Použité datové struktury

V mé práci se vyskytují tyto datové struktury:

3.4.1 B-strom

B-strom [1] je speciální případ (a,b)-stromu [17], který poskytuje větší volnost ve volbě minimálního a maximálního počtu potomků než B-strom.Autoři algoritmu, Rudolf Bayer a Ed McCreight, nikdy nevysvětlili, co v názvu znamená písmeno B. Nejčastěji se předpokládá, že znamená balanced.

Základní myšlenka B-stromu spočívá v tom, že uzly stromu mohou obsahovat n až $2n$ klíčů – počet položek ve stromu. Potom složitost operace vyhledávání v takovémto stromu bude v nejhorším případě řádu $log_n(N)$, kde N je po-čet položek ve stromu. Dalším důležitým hlediskem je faktor využití paměti který v případě B-stromu je minimálně 50 procent. To vše při zachování relativně malé složitosti operací potřebných na údržbu této struktury.

3.4.2 R-strom

R-strom (R-tree) je prostorová datová struktura navržená Antoninem Guttmanem v roce 1984. Ve své podstatě představuje R-strom jednoduchou modifikaci B-stromu. R-strom [8] je založen na shlukování v prostoru blízkých bodů na stejné stránky. V případě R-stromů jsou pro body konstruovány tzv minimální ohraničující obdelníky (angl. MBB –minimal bounding box), které shlukují podobné, blízké body. Body jednoho MBB jsou uloženy v listovém uzlu. Vnitřní uzly obsahující hierarchii MBB, tedy jakési nad-MBB. U R-stromů je garantován 50% faktor využití stránek v nejhorším případě, jako je tomu u B-stromů.

3.4.3 Sekvenční pole

Sekvenční pole, neboli stránkovaný seznam [8] je podle Piotra Wróblewského datová struktura, která je paměťově velice úsporná a umožňuje nám uspořádání libovolného počtu prvků. Počet prvků je omezen pouze velikostí disku. Oproti stromové datové struktuře se jedná se o lineární stránkovanou datovou strukturu, která je o poznání méně efektivnější než zmíněný B-strom. Složitost vyhledávání je zde lineární v závislosti na počtu prvků. Sekvenční pole obsahuje stránky (bloky), ve kterých jsou uloženy samotné záznamy. Každý blok má jedinečné id (pořadí bloku) a obsahuje ukazatel na následující blok.

4 Systémový katalog databáze

Systémový katalog [13] (Datový slovník) je nezbytnou součástí databáze. Uvnitř databáze jsou informace o databázových objektech, které se ukládají do tohoto katalogu ve formě pohledů. V podstatě je systémový katalog množina objektů, které obsahují informace které definují:

Systémový katalog může být rozdělen do logických skupin objektů. Toto řešení nabízí rozdělit například tabulky do skupiny, které jsou přístupné jen administrátorům a tabulky, které jsou přístupné i všem ostatním uživatelům. Například, uživatelé budou chtít vidět specifické práva databáze které obdrželi, nicméně nesplňují požadavky co se týče procesů databáze nebo vnitřní struktury.

Uživatel si většinou prohlédne katalog aby získal informace týkající se uživatelova vlastních objektů a privilegií, zatímco administrátor musí být schopný rozpoznat jakoukoliv událost nebo strukturu uvnitř databáze. V některých případech jsou vytvořeny objekty, které jsou přístupné jen administrátorům databáze.

Systémový katalog je extrémně důležitý pro administrátory databáze nebo pro všechny uživatele databáze, kteří si přejí pochopit strukturu a funkce databáze. Systémový katalog umožňuje udržovat pořádek nejen uživateli a správci, ale i databázovému serveru.

4.1 Tabulky systémového katalogu existujících databázových systémů

Kniha databázové systémy [8] říká: Standard SQL92 pohledy systémového katalogu definuje, nicméně standard byl přijat až poté co výrobci do svých implementací zavedli vlastní tabulky. V následující tabulce uvedeme názvy pohledů s podobným významem pro SQL standart SQL92 a Oracle.

Tabulka 3: Porovnání pohledů systémového katalogu

Standart SQL92	**Oracle**	**popis**
TABLES	`USER_TABLES`	Obsahuje jeden řádek pro každou tabulku kterou vlastní aktuální uživatel.
	`ALL_TABLES`	Obsahuje jeden řádek pro každou tabulku ke které má aktuální uživatel alespoň jedno přístupové právo.
COLUMNS	`USER_TAB_COLUMNS`	Obsahuje jeden řádek pro každý sloupec tabulky kterou vlastní aktuální uživatel
	`ALL_TAB_COLUMNS`	Obsahuje jeden řádek pro každý sloupec tabulky ke které má aktuální uživatel alespoň jedno přístupové právo.
USERS	`ALL_USERS`	Obsahuje jeden řádek pro každého uživatele v systému.
VIEWS	`USERS_VIEW`	Obsahuje jeden řádek pro každý pohled který vlastní aktuální uživatel.
	`ALLVIEW`	Obsahuje jeden řádek pro každý pohled ke kterému má aktuální uživatel alespoň jedno přístupové právo.
`TABLE_PRIVILEGES`	`USER_TAB_PRIVS`	Obsahuje jeden řádek pro každé právo přidělené uživatelem.
	`ALL_TAB_PRIVS`	Obsahuje jeden řádek pro každé právo kde aktuální uživatel je vlastníkem, dárcem nebo příjemcem.

5 Embedded databázový systém RadegastDB

Jedním z dalších embedded databázových systémů je RadegastDB, který je vyvíjen výzkumnou skupinou Katedry informatiky, Fakulty elektrotechniky a informatiky, VŠB – Technické univerzity Ostrava. Tento databázový systém je implementován v jazyce C++ a momentálně je stále ve vývoji. Databázový systém RadegastDB obsahuje sadu projektů, představujících datové struktury, běžně používaných v rámci dnešních relačních DBMS, a také prototyp nativní XML databáze. Každý projekt obsahuje validační test, kterým se ověřuje funkčnost a měření propustnosti jednotlivých operací. Tento test je vykonáván formou konzolové aplikace. [2]

5.1 Využité datové struktury

Každá datová struktura systému RadegastDB se skládá z hlavičky, která se po vytvoření ukládá do metadat a díky tomu jsme schopni kdykoliv znovu modifikovat tuto strukturu. Dále se pak skládá z instance samotné struktury. V tabulce 4 jsou zobrazeny použité struktury, jejich hlavičky a instance struktury.

Tabulka 4: Použité datové struktury RadegastDB

Datová struktura	**Hlavička**	**Třída reprezentující instanci**
Sekvenční pole	cSequentialArrayHeader	cSequentialArray
R-strom	cRTreeHeader	cRTree
B^+-strom	cBpTreeHeader	cBpTree

5.2 Datové typy a space descriptor systému RadegastDB

Systém obsahuje několik důležitých datových typů. Nejvíce jsem ve své práci využíval datové typy *cTuple*, *cNTuple* a *cHNTuple*.

cTuple představuje entici dat primitivních datových typů tedy int,short atd. nebo entici homogenních datových typů. Velikost entice nesmí být proměnné délky.

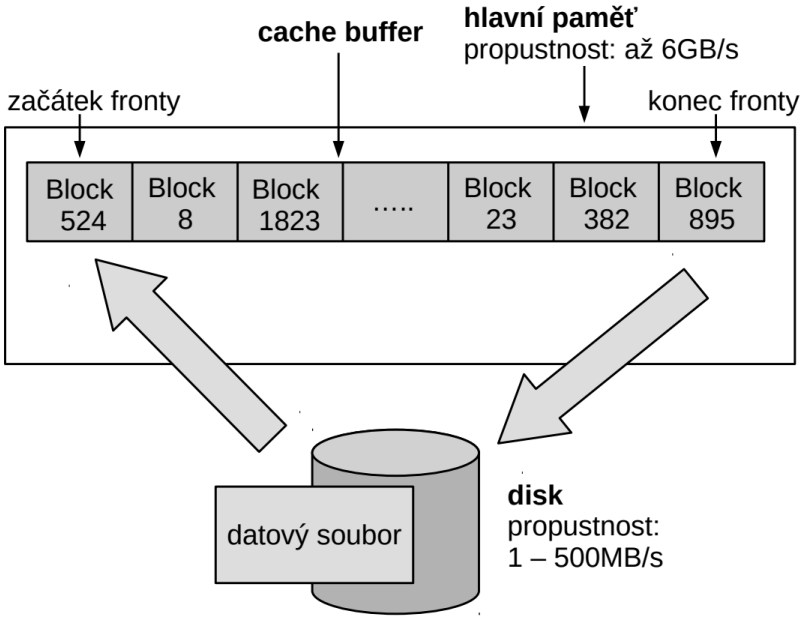
cNTuple představuje v systému VARCHAR, tedy entici znaků které jsou proměnné délky. Tato délka je uložena na prvním bajtu *cNTuple*.

cHNtuple představuje entici kombinace „jednoduchých“ datových typů a složitějších, jako je třeba VARCHAR. Délka entice je uložena na prvním bajtu *cHNTuple*.

cSpaceDescriptor slouží k identifikaci charakteru vícerozměrných dat entic tzn. říká nám, jaké datové typy obsahuje a velikost dimenze, což je počet prvků v entici. Také určuje velikost klíče datové struktury.

5.3 Cache Buffer

Databázové systémy obsahují v hlavní paměti cache buffer pro zajištění perzistence a výměny stránek stránek mezi diskem a hlavní pamětí. Na obrázku 2 je ukázka cache bufferu databázového systému.



Obrázek 2: Ukázka cache bufferu

Základní strukturu systému RadegastDB tvoří třída cQuickDB [20], která uchovává instance tříd cNodeCache, cMemoryPool a stará se o stránkování struktur. Instance této třídy je použita při vytváření datových struktur. Třída cNodeCache umožňuje načítání jednotlivých uzlů datových struktur do operační paměti, což zrychluje práci s nimi. Všechny operace s datovými strukturami jsou prováděny v rámci operační paměti. Tato třída dále zajišťuje zamykání načtených uzlů aby byla zajištěna konzistence datových struktur. Třída cMemoryPool slouží jako pomocná paměť při provádění operací na datovými strukturami. Tato třída je stejně jako cNodeCache sdílená pro celý databázovém systému.

6 Analýza implementace nástavby JDD

6.1 Překladač JDD

Aplikace využívá dva druhy JDD překladače. Jeden reaguje na příkaz CREATE TABLE, tedy příkaz na vytvoření nové tabulky a druhý na CREATE INDEX, který vytvoří jen index na existující tabulku. Překladač je schopen přeložit datové typy int, unsigned int, char, float, double a string, který je v SQL reprezentován jako varchar.

6.2 Zápis do systémového katalogu

K řešení toho problému, jsem využil knihovnu serializace knihovny Boost [4]. Vybral jsem ji kvůli odladěné implementaci a jednoduchému použití. Zápis se prování do souboru SystemCatalog.dat. Do mého systémového katalogu zapisuji tyto údaje :

- Jméno indexu,
- jméno tabulky,
- typ datové struktury - typ datové struktury indexu,
- sloupce tabulky - vektor sloupců tabulky,
- data za klíčem proměnné délky - informace, zda jsou data za klíčem proměnné délky,
- klíč proměnné délky - informace, zda je klíč proměnné délky,
- homogenní - informace, zda je tabulka homogenní,
- index na primární klíč - informace, jestli se jedná o index primárního klíče,
- data tabulky proměnné délky - informace, zda jsou celé záznamy tabulky proměnné délky,
- klíčové sloupce - vektor sloupců, které jsou uloženy v klíči,
- pozice klíče - vektor pozic klíče v tabulce.

Jméno indexu ukládám kvůli rozlišení jednotlivých indexů. Jméno tabulky ukládám kvůli identifikace, s jakými indexy které tabulky budu pracovat. Data za klíčem proměnné délky, klíč proměnné délky a typ datové struktury ukládám, protože u otevírání hlaviček tyto informace potřebujeme. Index na primární klíč ukládám, abych mohl rozeznat index primárního klíče tabulky od ostatních indexů tabulky. Data tabulky proměnné délky ukládám jsou potřebné při práci se záznamy shlukované tabulky.

7 Fyzická implementace databázových systémů

Fyzická implementace [8] definuje datové struktury a způsob uložení pro základní logické objekty: uložení tabulek, typy indexů (tímto se zabývá má práce) a materializované pohledy. Fyzická implementace tedy řeší uložení dat na nejnižší úrovni databáze. Dostupných datových struktur je celá řada, nabízí databázový systém administrátorovi nějakou volbu.

7.1 Fyzický návrh

Při zadání příkazu, který obsahuje prefix CREATE TABLE, se v databázi vytváří tabulka typu haldy s indexem typu B-strom (nebo R-strom), nebo shlukovaná tabulka typu B-strom (nebo R-strom). Při zadání příkazu CREATE INDEX se v databázi vytvoří nový index buď typu B-strom nebo R-strom, záleží na typu datové struktury primárního klíče tabulky.

Aplikace je schopna vytvářet dva druhy klíčů a to jednoduchý a složený. Jednoduchý klíč obsahuje jen jednu klíčovou hodnotu, avšak složený klíč klíčových hodnot obsahuje více. Pro tabulky typu halda platí, že za klíčovými hodnotami jejich indexu je uloženo ROWID. ROWID je dvojice id uzlu sekvenčního pole představující haldu a pořadí v tomto uzlu.

Používají se dva druhy dotazu a to bodový a rozsahový dotaz. Bodový dotaz vrací jen jeden záznam a to záznam se stejnou hodnotou klíče, který je zadán v parametru dotazu. Rozsahový dotaz vrací všechny záznamy v rozsahu minimální hodnoty klíče až maximální hodnoty klíče zadané v parametru dotazu.

7.2 Použité vzory fyzického návrhu

V této práci byly použity 4 vzory. Tři ze čtyř těchto vzorů reagují na příkaz CREATE TABLE. Rozdíl je v tom, že pro každý tento vzor se vytvoří jiný typ struktury. Posledním vzorem je reakce na příkaz CREATE INDEX.

- Vzor 1: CREATE TABLE (B-tree) - Jedná se o vytvoření tabulky typu halda s indexem typu B-strom pro primární klíč. Tento vzor je běžně implicitní možností při vytváření tabulky.
- Vzor 2: CREATE TABLE (R-tree) - Jedná se o vytvoření tabulky typu halda s indexem typu R-strom pro primární klíč. U R-stromu se vytváří jen složené indexy, pro jeden atribut to není vhodné.
- Vzor 3: CREATE TABLE (shlukovaná tabulka) - Jedná se o vytvoření tabulky typu shlukovaná tabulka s indexem typu B-strom nebo R-strom pro primární klíč, záleží na volbě uživatele.
- Vzor 4: CREATE INDEX - Tento vzor nevytváří tabulku, jen index na existující tabulku. Po vytvoření indexu jsou automaticky vložena data z tabulky. Implicitní možností pro tento vzor je index typu B-strom.

7.2.1 Rozdíl mezi dvojicí index, halda a shlukovanou tabulkou

Rozdíl mezi dvojicí index a halda a shlukovanou tabulkou je v obsahu indexu. U vzorů 1 a 2, index obsahuje klíč, což je hodnota indexovaného atributu a ROWID, které obsahují vždy dvě hodnoty a to číslo uzlu sekvenčního pole a pozici v tomto uzlu. Nicméně ve vzoru 3 halda neexistuje, klíč se skládá z hodnoty klíče a dat, ve kterých je uložen zbytek indexovaného záznamu. Z toho vyplývá, že proces vytváření indexů na již existující tabulku, která obsahuje data, se bude lišit.

7.3 Základní plány vykonání dotazu

Plán vykonání dotazu nám říká, co se děje, když se přeloží zadaný dotaz.

```
SELECT * from student where name='Honza'
```

Výpis 1: Přikaz SELECT

Pokud je tabulka student typu halda, Oracle provede TABLE ACCES(FULL). Pokud bychom měli vytvořený index typu B-strom na atribut name, tak se bude provádět operace UNIQUE SCAN na index tabulky a následně by se provedl TABLE ACCESS BY INDEX ROWID na tabulku typu halda.

7.4 Příklad vytvoření struktury pomocí systému RadegastDB

V následujícím úryvku kódu je předvedenou, jak v praxi vypadá vytváření indexu typu B-strom pomocí pomocí systému RadegastDB.

```
cQuickDB *quickDB = new cQuickDB();//instance databaze
if (!quickDB->Open(dbPath, CACHE_SIZE, MAX_NODE_INMEM_SIZE, BLOCK_SIZE))//
    otevreni existujici databaze
   {
   printf("Critical Error: Cache Data File was not open!\n");
   exit(1);
   }
mIndexHeader = new cBpTreeHeader<cTuple>(uniqueName,BLOCK_SIZE, dd, keySize,
    dataSize, variableLenData, DSMODE, cDStructConst::BTREE, COMPRESSION_RATIO)
    ;//vytvoreni hlavicky datove struktury pomoci konstant
mIndexHeader->SetRuntimeMode(RUNTIME_MODE);
mIndexHeader->SetCodeType(CODETYPE);
mIndexHeader->SetHistogramEnabled(HISTOGRAMS);
mIndexHeader->SetInMemCacheSize(INMEMCACHE_SIZE);

mIndex = new cBpTree<cTuple>();
```

```
if (!mIndex->Create(mIndexHeader, quickDB))//vytvoreni indexu typu B-strom
{
   printf("Key index: creation failed!\n");
   return false;
}
else
   return true;

quickDB->Close();//uzavreni databaze, pri kterem se vytvorena struktura ulozi
    do databaze
```

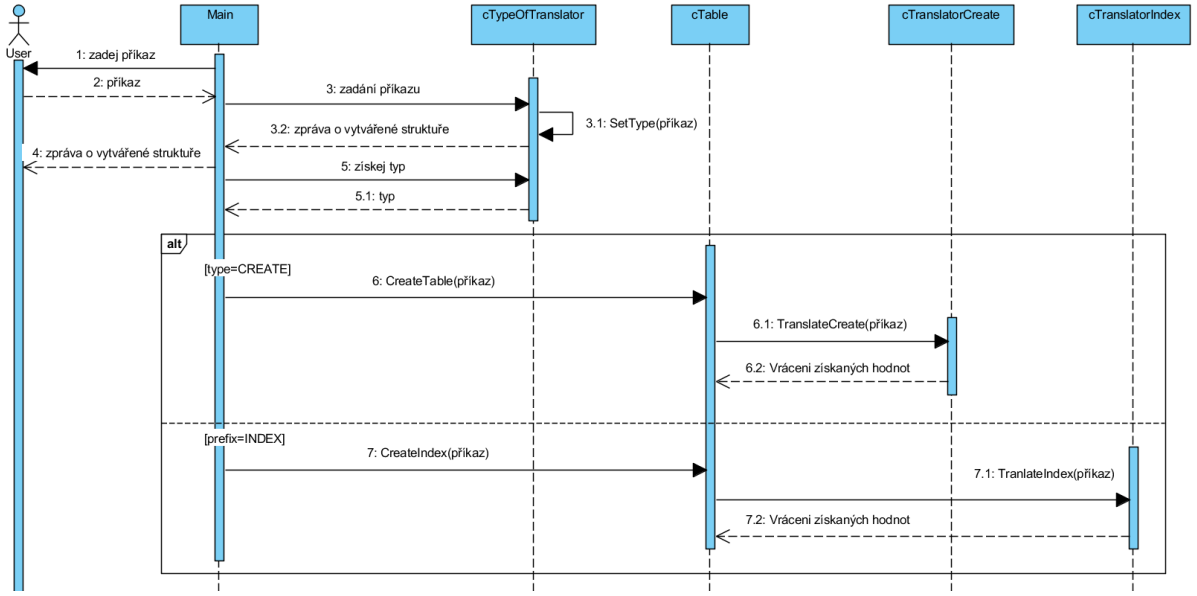
Výpis 2: Vytvoření struktury typu B-strom pro klíč proměnné délky

8 Implementace

V této části bude popsána implementace vrstvy a popsány použité vzory.

8.1 Postup vytváření jednotlivých vzorů

Všechny vzory 7.2 mají společný začátek při jejich vytváření a to přečtení prefixu příkazu uživatele pomocí třídy *cTypeOfTranslator*. Po vyhodnocení, o jaký typ příkazu se jedná, jestli CREATE TABLE nebo CREATE INDEX se vytvoří instance třídy *cTable* a ta zavolá podle výsledného prefixu buď metodu `CreateTable` nebo `CreateIndex`. Tento postup je popsán na obrázku 3.



Obrázek 3: Sekvenční diagram: Rozpoznání typu příkazu

8.1.1 Vzor 1

Vzor 1 je použit pokud se volá metoda `CreateTable` a překladač JDD *cTranslatorCreate*, který je volán uvnitř této metody najde na konci příkazu výraz OPTION:BTREE, nebo nenajde žádný výraz OPTION. Jedná se tedy o defaultní konstrukci. Po vyhodnocení vrátí překladač všechny potřebné proměnné pomocí metody `TranlateCreate` pro vytvoření dané datové struktury. Po zpracování těchto proměnných metodou `CreateTable` se vytvoří hlavička datové struktury typu B-strom *cBpTreeHeader* a ta vytvoří samotnou instanci B-stromu *cBpTree*. Následně se tato datová strukturu z testovacích důvodů plní daty a nakonec se uloží do systémového katalogu.

8.1.2 Vzor 2

Vzor 2 je použit pokud se volá metoda `CreateTable` a překladač JDD *cTranslatorCreate*, který je volán uvnitř této metody najde na konci příkazu výraz `OPTION:MD_TABLE`. Po vyhodnocení vrátí překladač všechny potřebné proměnné pomocí metody `TranlateCreate` pro vytvoření dané datové struktury. Po zpracování těchto proměnných metodou `CreateTable` se vytvoří hlavička datové struktury typu B-strom *cRTreeHeader* a ta vytvoří samotnou instanci B-stromu *cRTree*. Následně se tato strukturu z testovacích důvodů plní daty a nakonec se uloží do systémového katalogu.

8.1.3 Vzor 3

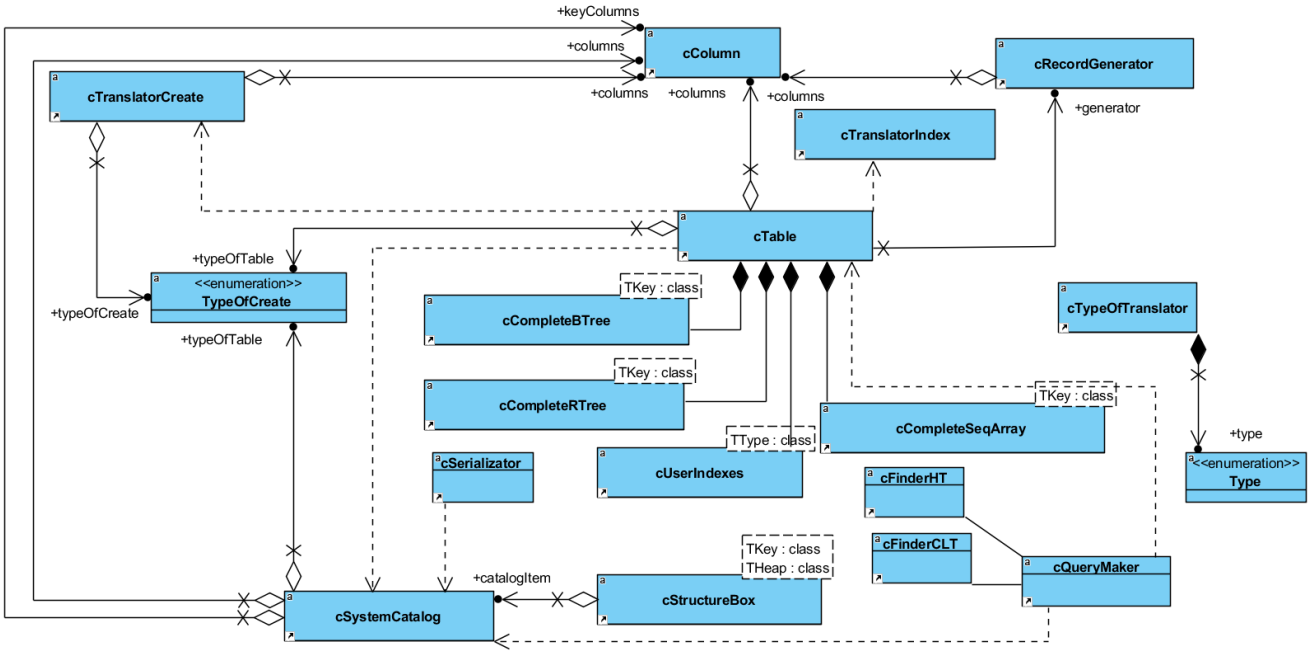
Vzor 3 je použit pokud se volá metoda `CreateTable` a překladač JDD *cTranslatorCreate*, který je volán uvnitř této metody najde na konci příkazu výraz `OPTION:CLUSTERED_TABLE` nebo `OPTION:CLUSTERED_TABLE(MD_TABLE)`. Po vyhodnocení vrátí překladač všechny potřebné proměnné pomocí metody `TranlateCreate` pro vytvoření dané datové struktury. Po zpracování těchto proměnných metodou `CreateTable` se rozhodne, zda se má vytvořit klíč typu B-strom nebo struktura typu R-strom. To se rozhoduje podle výrazu na konci příkazu, `CLUSTERED_TABLE` je B-strom a `OPTION:CLUSTERED_TABLE(MD_TABLE)` je R-strom. Zbytek procesu je stejný, jako u předešlých dvou vzorů.

8.1.4 Vzor 4

Vzor 4 jako jediný reaguje na prefix `CREATE_INDEX`, kde instance *cTable* volá metodu `CeateIndex`. Ta obsahuje volání JDD překladače *cCreateIndex*, který z příkazu získá všechny potřebné data a zpracuje je. Podle těchto dat se pak vytváří příslušná datová struktura. Typ datové struktury se určuje z výrazu na konci příkazu. Výraz `OPTION:BTREE` vytvoří index typu B-strom a výraz `OPTION:MD_TABLE`. Vytváření indexu se provádí v metodě `ConstructIndex` (pro tabulku typu halda) nebo v metodách `ConstructClusteredIndexB` (shlukovanou tabulku typu B-strom) a `ConstructClusteredIndexB` (shlukovanou tabulku R-strom). V těchto metodách se vytvoří příslušná datová struktura a následně se indexují již existující data buď podle haldy nebo podle primárního klíče (shlukovaná tabulka).

8.2 Třídní diagram

Na obrázku 4 se nachází třídní diagram.



Obrázek 4: Třídní diagram

8.3 Popis důležitých tříd a jejich metod

- Třída *cColumn*: Tato třída představuje jeden sloupce v tabulce.
- Třída *cTypeOfTranslator*: Tato třída slouží k rozlišení, zda je vstup příkaz typu CREATE TABLE nebo CREATE INDEX. Její jediná proměnná s názvem type je typu enum TypeOfCreate (BTREE, RTREE, `CLUSTERED_TABLE_BTREE`, `CLUSTERED_TABLE_RTREE`), který rozlišuje prefix záznamu. Metoda SetType přečte prefix uživatelova vstupu a podle toho nastaví hodnotu type, se kterou se pak dále pracuje.
- Třída *cTranslatorCreate*: Tato třída představuje překladač pro příkaz CREATE TABLE. Datový typ int je namapován na výraz INT, unsigned int na BIGINT, short na SMALLINT, char na CHAR, float na FLOAT, double na DOUBLE a string na VARCHAR. Metoda `TranslateCreate` s parametrem typu string, který představuje vstup uživatele, prochází tento parametr v cyklu do-while a získá všechny potřebné proměnné. K tomu využívám hlavně metody string::find, string::insert a moji metodu `GetType`, která podle zadaného řetězce datového typu rozluští, o jaký datový typ databázového systému RadegastDB se jedná. Dále pak má metody `SetStructureType`, která zjistí, o jakou datovou strukturu se jedná a podle toho nastaví instační proměnnou typeOfCreate, a CheckDuplicity, která kontroluje, zda se v jedné tabulce nevyskytují sloupce se stejným jménem. Pokud existuje, tak vypíše chybovou hlášku a struktura se nevytvoří.
- Třída *cTranslatorIndex*: Tato třída představuje překladač pro příkaz CREATE INDEX. Metoda `TranslateCreateIndex`, jejíž parametrem je vstup uživatele prochází v cyklu

vstup uživatele,kontroluje syntaxi SQL příkazu CREATE INDEX a plní instanční proměnné získanými daty. Syntaxí je tato třída velmi podobná metodě `TranslateCreate` třídy *cTranslatorCreate*.

- Třída *cRecordGenerator*: Tato třída slouží ke generování náhodných záznamu. Tato třída pomocí svých metod a předaných parametrů space descriptoru tabulky, pro kterou generují data a všech sloupců tabulky, generuje data podle datových typů sloupců. To se děje v metodě CreateNewRecord. Třída také rozlušije, jestli může generovat NULL hodnotu a pokud ano, tak nastaví hodnotu na maximální hodnotu datového typu sloupce a při výpisu se takový záznam vypisuje jako NULL.

- Třída *cStructureBox*: Tato třída představuje data class pro instance indexu typu B-strom a R-strom, haldy tabulky a záznam systémového katalogu příslušného indexu.

- Třída *cSystemCatalog* - Tato třída představuje jeden záznam v systémovém katalogu. Třída *cSystemCatalog* je serializovatelná, implementuje metodu `serialize` knihovny *Boost*, díky které instanci *cSystemCatalog* serializuji do systémového katalogu.

- Třída *cSerializator*: Tato třída slouží k serializaci a deserializaci dat ze systémového katalogu. Obsahuje dvě důležité statické metody. První je `Deserialize`. Tato metoda pomocí knihovny *Boost* deserializuje ze souboru SystemCatalog.dat všechny existující záznamy v systémovém katalogu a převede ho na vektor typu *cSystemCatalog*, který představuje návratovou hodnotu funkce. Druhá je Serialization. V této metodě se nejdříve zavolá metoda `Deserialize`, která vrátí vektor typu *cSystemCatalog*. Její parametry jsou použity k vytvoření instance *cSystemCatalog*, která je následně vložena na konec deserializovaného vektoru typu *cSystemCatalog*, který je následně celý serializován pomocí knihovny *Boost*.

- Třída *cTable*: Tato třída představuje středobod celé aplikace. Třída *cTable* reprezentuje instanci tabulky. Metoda `CreateTable` slouží k vytváření nových tabulek. V ní se příkaz uživatele přeloží pomocí překladače *cTranslateCreate*. Důležitou částí metody `CreateTable` je deserializace systémového katalogu do vektoru a kontrola, jestli už tabulka se stejným jménem neexistuje. Podle těchto dat následně metoda nastaví všechny instanční proměnné, které potřebujeme, jako je třeba space descriptor pro celý záznam, space descriptor klíče tabulky, sloupce tabulky atd. Pomocí těchto dat potom vytvoříme požadovanou strukturu nebo struktury, záleží na typu vzoru. Na konec se potřebné proměnné uloží do instance *cSystemCatalog*, která se vloží na konec deserializovaného vektoru a tento vektor se serializuje do systémového katalogu.

  Metoda `CreateIndex`, slouží k tvorbě samostatných indexů na již existující tabulky. Tato metoda pomocí překladače JDD *cTranslatorIndex* zjistí vše potřebné k tomu, aby byl index vytvořen, včetně jména tabulky. Toto jméno tabulky hledáme v deserializovaném vektoru typu *cSystemCatalog*. Když ho najdeme, všechny proměnné z toho záznamu vy-

táhneme a znovuvytvoříme podle nich požadovanou tabulku. Následně vytváříme požadovanou strukturu v metodách `CreateIndex` (pro tabulku typu halda) nebo metodách `ConstructClusteredIndex`. Po vytvoření se v těchto metodách musejí indexovat již existující data podle primárního klíče (pro shlukovanou tabulku) nebo haldy tabulky (pro tabulku typu halda). Abychom otevřeli index primárního klíče, musíme otevřít všechny datové struktury, které jsou v databázi uložené (z důvodu funkcionality systému RadegastDB). K tomu aplikaci slouží systémový katalog, metody `OpenBTreeHeader` a `OpenRtreeHeader` (pro tabulku typu halda) a nebo metody `OpenClusteredBHeader` a `OpenClusteredBHeader` (pro shlukovanou tabulku). K těmto indexům posléze přiřadíme příslušné záznamy ze systémového katalogu a zabalíme do balíku typu *cUserIndexes* a vložíme do vektoru indexů pomocí metody *FillUserIndexes*, pokud se jedná primární klíč, tak ten se ukládá do instanční proměnné typu *cCompleteBTree* nebo *cCompleteRTree* pomocí metod `FillKeyIndexes`. Pokud se jedná o tabulku podle vzoru 1 nebo vzoru 2, tak halda se rovněž otevírá a následně ukládá do instanční proměnné typu *cCompleteSeqArray*.

Vkládání nových hodnot do indexu se provádí v metodách *SetValueCT* (pro vzor 3) a *SetValueHT (pro vzor 1 a 2)*, kde se nejprve vloží příslušná hodnota do indexu, který představuje primární klíč pomocí metod *InsertToKeyClusteres* (pro vzor 3) nebo *InsertToHTIndexB* a *InsertToHTIndexR* (pro vzor 1 a 2). Potom se kontroluje, jestli tabulka má více indexů. Pokud ano tak se do těchto indexů vkládají hodnoty pomocí *InsertCTIndexesBtree* a *InsertCTIndexesRtree* nebo *InsertIndexesBtree* a *InsertIndexesRtree*.

- Třída *cFinderHT*: Úkolem této třídy je vytvoření dotazu a zpracování dotazu pro indexy typu vzor 1 a vzor 2. Obsahuje metodu `FindB`, kde se provádí rozsahový dotaz na index typu B-strom, který chceme dotazovat. Tento rozsahový dotaz vrací ukazatel na klíč v indexu, za kterým jsou jeho data a následně další klíče. V datech je uloženo ROWID 7.1, takže při získání dat, si pomocí těchto dvou proměnných vytáhneme z haldy celý záznam a provedeme výpis. Na další klíče se dostaneme pomocí metody `MoveBuffer`. Metoda `FindR` je principiálně stejná, ale metoda rozsahového dotazu pro R-strom má jiné parametry.

- Třída *cFinderCT*: Jejím úkolem je vytvoření dotazu a zpracování dotazu pro indexy shlukované tabulky. Obsahuje metodu `FindB`, kde se provádí rozsahový dotaz na index typu B-strom, který chceme dotazovat. Stejně jako u metod třídy `cFinderHT` je vracen ukazatel na klíč a data, nicméně shlukovaná tabulka nemá haldu, ale na kompletní záznam ukazuje ukazatel rozsahového dotazu. Z toho vyplývá, že nejprve získáme a vypíšeme klíč, posunem se na data klíče pomocí metody `ShiftKey`, získáme a vypíšeme data a posuneme se pomocí metody `ShiftData` na další klíč. Metoda `FindR` je principiálně stejná, ale metoda rozsahového dotazu pro R-strom má jiné parametry.

9 Testování jednotlivých vzorů

9.1 Úvod testování jednotlivých vzorů

Pro testování a porovnání vzorů CREATE TABLE použijeme jeden obecný SQL příkaz, který však budeme používat pro různé datové struktury. U R-stromu navíc bude složený primární klíč, jelikož je R-strom vícerozměrná datová struktura, jeho použití jako indexu pro jeden atributu není vhodné. Pro otestování vzoru CREATE INDEX bude vytvářet pro jednotlivé vzory stejný uživatelský index. Jediný rozdíl bude v názvech indexů a tabulek. Do každé vytvořené tabulky bude vloženo 100 000 záznamů. Při druhém vkládání bude vloženo dalších 100 000 záznamů.

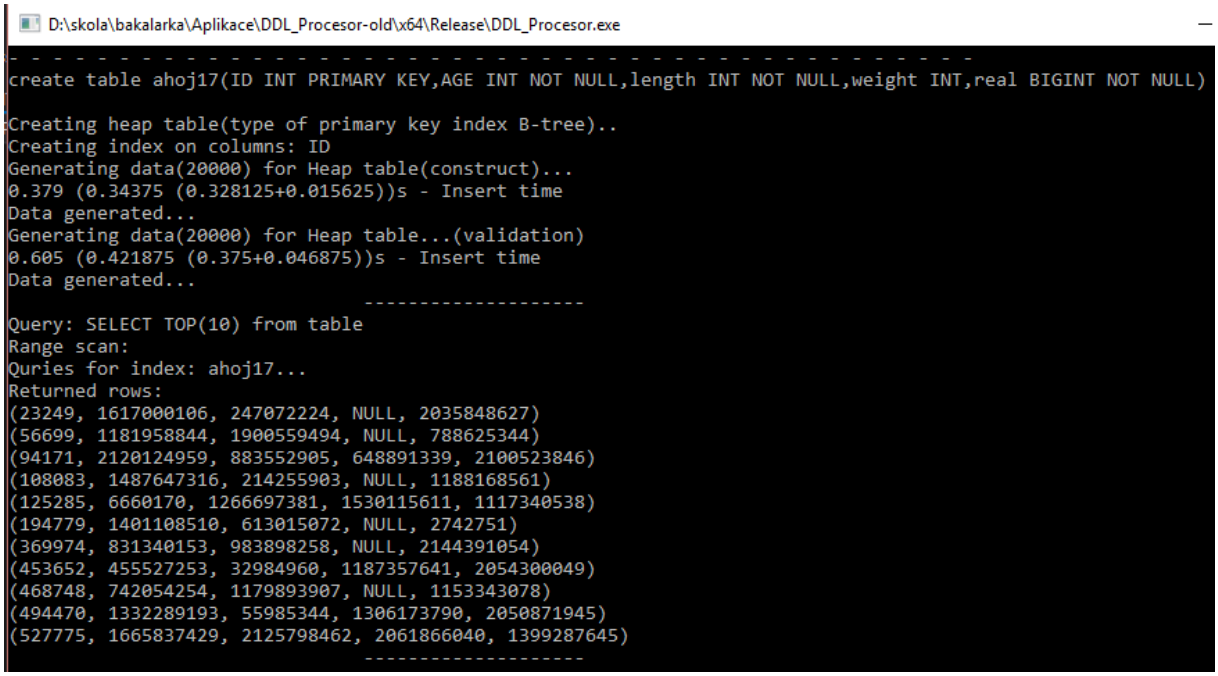
```
CREATE TABLE MyTableM(column1 INT PRIMARY KEY,column2 INT NOT NULL,column3
    FLOAT,column4 INT NOT NULL,column5 CHAR NOT NULL,column6 BIGINT NOT NULL)
```

Výpis 3: Obecný příkaz CREATE TABLE

Průběh testování:

- V případě příkazu CREATE TABLE otevření databáze, vytvoření nové datové struktury představující tabulku typu halda nebo typu shlukovaná tabulka a naplnění daty z důvodu validace.
- V případě příkazu CREATE INDEX otevření databáze, vytvoření nové datové struktury na existující tabulku a indexace existujících dat.
- Uzavření databáze a uložení vytvořené struktury do systémového katalogu.
- Znovu otevření databáze a všech struktur tabulky.
- Druhé plnění tabulky daty a také všech indexů tabulky.
- Dotazování posledního vytvořeného indexu tabulky.
- Výpis výsledku dotazu a uzavření databáze.

Na obrázku 5 vidíme výpis na obrazovku pro vzory 1,2 a 3, u B-stromu a R-stromu se liší druh indexu a forma dotazu.



Obrázek 5: Vzorový výpis vzorů 1, 2 a 3

9.2 Vzor CREATE TABLE (B-tree)

Pro tento vzor by příkaz vypadal takto:

```
CREATE TABLE MyTableM(column1 INT PRIMARY KEY,column2 INT NOT NULL,column3
    FLOAT,column4 INT NOT NULL,column5 CHAR NOT NULL,column6 BIGINT NOT NULL)
    OPTION:BTREE
--nebo
CREATE TABLE MyTableM(column1 INT PRIMARY KEY,column2 INT NOT NULL,column3
    FLOAT,column4 INT NOT NULL,column5 CHAR NOT NULL,column6 BIGINT NOT NULL)
```

Výpis 4: Příkaz CREATE TABLE vzor 1

V tomto případě se nám vytvoří halda typu *cSequentialArray<cTuple>* 5.1 a index na primární klíč typu *cBpTree<cTuple>* 5.1. Hned po vytvoření probíhá první generování, uzavření databáze a uložení do systémového katalogu. Jako další krok se provede znovuotevření poslední vytvořené struktury a provede se druhé generování záznamů. Nakonec se dotazujeme na vytvořenou datovou strukturu a vracíme výsledek.

První vkládání u toho vzoru trvalo 2,2 s. Po znovuotevření databáze druhé vkládání trvalo 2,5 s. Celková doba vkládání dat je tedy 4,7 s. Velikost databáze kde je uložena tabulka a B-strom je 4,85 MB. Velikost systémového katalogu je 1 kB.

9.3 Vzor CREATE TABLE (R-tree)

Pro tento vzor by příkaz vypadal takto:

```
CREATE TABLE MyTableM(column1 INT PRIMARY KEY,column2 INT NOT NULL,column3
    FLOAT,column4 INT NOT NULL,column5 CHAR PRIMARY KEY,column6 BIGINT NOT NULL
    ) OPTION:MD_TABLE
```

Výpis 5: Příkaz CREATE TABLE vzor 2

V tomto případě se nám vytvoří primární klíč (index) typu *cRTree<cTuple>* 5.1 a halda typu *cSequentialArray<cTuple>*. Hned po vytvoření probíhá první generování, uzavření databáze a uložení do systémového katalogu. Jako další krok se provede znovuotevření poslední vytvořené struktury a provede se druhé generování záznamů. Nakonec se dotazujeme na vytvořenou datovou strukturu a vracíme výsledek.

První vkládání u toho vzoru trvalo 2,8 s. Po znovuotevření databáze druhé vkládání trvalo 2,8 s. Celková doba vkládání dat je tedy 5,7 s. Velikost databáze kde je uložena tabulka a R-strom je 6,9 MB. Velikost systémového katalogu je 1 kB.

9.4 Vzor CREATE TABLE (shlukovaná tabulka)

V tomto případě se vytvoří požadovaná struktura (index typu B-strom nebo index typu R-strom),do které se vloží klíč a data. Na rozdíl od vzoru 1 a vzoru 2, data u tohoto vzoru tvoří zbytek záznamu, tudíž nepotřebujeme haldu.

Pro tento vzor s indexem typu B-strom by příkaz vypadal takto:

```
CREATE TABLE MyTableM(column1 INT PRIMARY KEY,column2 INT NOT NULL,column3
    FLOAT,column4 INT NOT NULL,column5 CHAR NOT NULL,column6 BIGINT NOT NULL)
    OPTION:CLUSTERED_TABLE
```

Výpis 6: Příkaz CREATE TABLE vzor 3: B-strom

Zde se vytvoří primární klíč (index) typu *cBpTree<cTuple>* s dostatečně velkým prostorem za klíčem pro data, který se určuje v hlavičce. Hned po vytvoření probíhá první generování, uzavření databáze a uložení do systémového katalogu. Jako další krok se provede znovuotevření poslední vytvořené struktury a provede se druhé generování záznamů. Nakonec se dotazujeme na vytvořenou datovou strukturu a vracíme výsledek.

První vkládání u toho vzoru trvalo 2,05 s. Po znovuotevření databáze druhé vkládání trvalo 2,5 s. Celková doba vkládání dat je tedy 4,6 s. Velikost databáze, kde je uložena shlukovaná tabulka typu B-strom je 6 MB. Velikost systémového katalogu je 1 kB.

Pro index typu R-strom by příkaz vypadal takto:

```
CREATE TABLE MyTableM(column1 INT PRIMARY KEY,column2 INT NOT NULL,column3
    FLOAT,column4 INT NOT NULL,column5 CHAR PRIMARY KEY,column6 BIGINT NOT NULL
    ) OPTION:CLUSTERED_TABLE(MD_TABLE)
```

Výpis 7: Příkaz CREATE TABLE vzor 3: R-tree

Zde se vytvoří primární klíč(index) typu *cRTree<cTuple>*. Zbytek procesu je stejný jako u indexu typu B-strom.

První vkládání u toho vzoru trvalo 2,5 s. Po znovuotevření databáze druhé vkládání trvalo 2,6 s. Celková doba vkládání dat je tedy 5,23 s. Velikost databáze, kde je uložena shlukovaná tabulka typu R-strom je 4,85 MB. Velikost systémového katalogu je 1 kB.

9.5 Vzor CREATE INDEX

Pomocí toho vzoru budeme vytvářet jednoduchý nebo složený index pro tabulku typu halda nebo shlukovanou tabulku. Máme tři varianty pro příkaz CREATE INDEX pro vytvoření složeného indexu dvou atributů. Pro vytvoření indexu typu B-strom používáme následující příkaz:

```
CREATE INDEX indexN ON MyTableM(column2,column4)
--nebo
CREATE INDEX indexN ON MyTableM(column2,column4) OPTION:BTREE
```

Výpis 8: Příkaz CREATE INDEX pro složený index typu B-strom (implicitní)

Pro vytvoření indexu typu R-strom používáme následující příkaz:

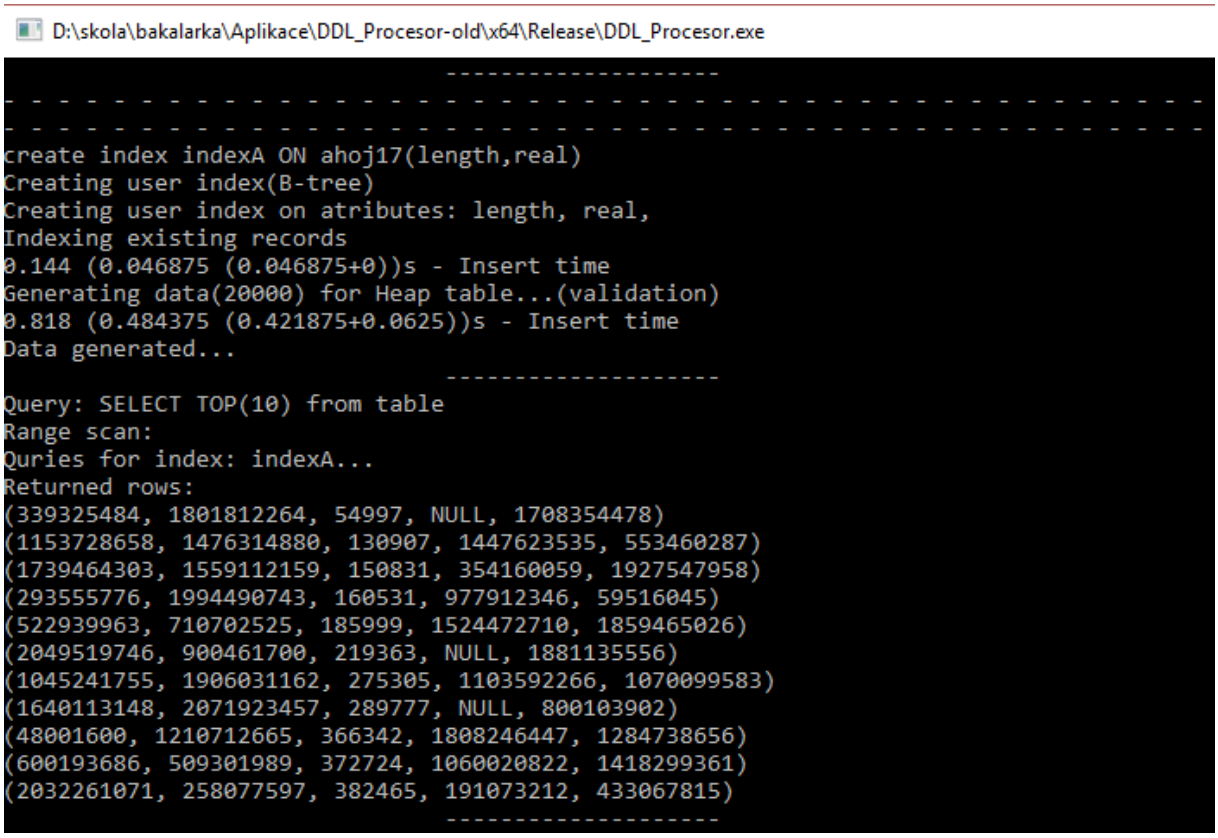
```
CREATE INDEX indexN ON MyTableM(column2,column4) OPTION:MD_TABLE
```

Výpis 9: Příkaz CREATE INDEX pro složený index typu R-strom

U všech datových struktur, se po vytvoření nejdříve indexují již existující data. Poté vkládáme nové záznamy jak do indexu primárního klíče MyTableM, tak do nového indexu indexN. Pro náš test budou indexy stejného typu, jako je typ primárního klíče tabulky.

U vzoru jedna 1 a 2 je průběh stejný. Prochází se sekvenčně halda, ze které pak jednotlivé záznamy indexujeme. U vzoru dva se volá na index rozsahový dotaz, který vrátí všechny klíče a jejich data. Následně příslušný sloupec indexujeme a zbytek dat ukládáme za klíč.

Na obrázku 6 se nachází vzorový výpis pro vzor 4.



Obrázek 6: Vzorový výpis vzoru 4

Při vytváření indexu na tabulku typu halda s indexem typu B-strom se existující data indexovaly 0,6 s a čas vkládání nových dat byl 4,2 s. Celkový čas indexování existujících dat a vkládání nových dat je tedy 4,8 s. Velikost databáze kde je uložena tabulka typu halda, index typu B-strom jako primární klíč a nový vytvořený index typu B-strom je 16,6 MB. Velikost systémového katalogu je 1 kB.

Při vytváření indexu na tabulku typu halda s indexem typu R-strom se existující data indexovaly 1,7 s a čas vkládání nových dat byl 3,9 s. Celkový čas indexování existujících dat a vkládání nových dat je tedy 5,6 s. Velikost databáze kde je uložena tabulku typu halda, index typu R-strom jako primární klíč a nový vytvořený index typu R-strom je 14,6 MB. Velikost systémového katalogu je 1 kB.

Při vytváření indexu na shlukovanou tabulku typu B-strom se existující data indexovaly 0,3 s a čas vkládání nových dat byl 6,6 s. Celkový čas indexování existujících dat a vkládání nových dat je tedy 7,8 s. Velikost databáze kde je uložena shlukovaná tabulka typu B-strom a nový vytvořený index typu B-strom je 18,1 MB. Velikost systémového katalogu je 1 kB.

Při vytváření indexu na shlukovanou tabulku typu R-strom se existující data indexovaly 1,4 s a čas vkládání nových dat byl 3,9 s. Celkový čas indexování existujících dat a vkládání nových

dat je tedy 5,4 s. Velikost databáze kde je uložena shlukovaná tabulka typu R-strom a nový vytvořený index typu R-strom je 13,6 MB. Velikost systémového katalogu je 1 kB.

9.6 Výsledky testů

V tabulce 5 se nachází výsledky testování vzorů. V tabulce 6 se nachází délka testování jednotlivých úseků testu.

Tabulka 5: Výsledky testování jednotlivých vzorů

Vzor	**Čas vkládání (s)**	**Velikost databáze (MB)**
Vzor 1	4,7	7,0
Vzor 2	5,7	6,9
Vzor 3 (B-strom)	4,6	6,0
Vzor 3 (R-strom)	5,2	4,8
Vzor 4 (B-strom) na tabulku typu halda	4,8	16,6
Vzor 4 (R-strom) na tabulku typu halda	5,6	14,6
Vzor 4 (B-strom) na shlukovanou tabulku	7,0	18,1
Vzor 4 (R-strom) na shlukovanou tabulku	5,4	13,6

Tabulka 6: Podrobný výpis času trvání jednotlivých úseků

Vzor	**Vkládání 1 (s)**	**Index. existujících dat (s)**	**Vkládání 2 (s)**
Vzor 1	2,2	–	2,5
Vzor 2	2,8	–	2,8
Vzor 3 (B-strom)	2,0	–	2,5
Vzor 3 (R-strom)	2,5	–	2,6
Vzor 4 (B-strom) na tabulku typu halda	–	0,6	4,2
Vzor 4 (R-strom) na tabulku typu halda	–	1,7	3,9
Vzor 4 (B-strom) na shlukovanou tabulku	–	0,3	6,6
Vzor 4 (R-strom) na shlukovanou tabulku	–	1,4	3,9

9.7 Porovnání s již existujícími databázovými systémy

V tabulce 7 jsou zobrazeny podporované struktury a funkce vybraných, již existujících embedded databázových systémů. Všechny uvedené struktury a funkce jsou podporovány systémem

RadegastDB a mou aplikací.

Tabulka 7: Srovnání embedded databázových systémů

Název	**BTree**	**RTree**	**Clustered table**	**JDD SQL**	**SQL dotazy**
Berkeley DB	Ano	Ne	Ne	Ano	Ano
SQL Lite	Ano	Ano	Ano	Ano	Ano
Firebird	Ano	Ne	Ne	Ano	Ano
SQL Server Compact	Ano	Ne	Ne	Ano	Ano
InnoDB	Ano	Ano	Ano	Ano	Ano
Mé řešení	Ano	Ano	Ano	Ano	Ano

9.8 Podporované datové typy embedded databázového systému RadegastDB

Z důvodu chybějící implementace na straně systému RadegastDB, jsou některé datové typy nebo jejich kombinace nepodporovány.

Moje aplikace nepodporuje vytváření R-stromu na tabulku, kde se vyskytuje sloupec proměnné délky (tudíž *cRTree<cNTuple>*, *cRTree<cHNTuple>* je nepodporován). Dále pak datové typy SMALLINT(*cShort*) a CHAR(*cChar*) nejdou nastavit jako primární klíče (ani jako kombinace pro složený index), protože jejich velikost je moc malá a systém s tak malou velikostí nepracuje. Z toho vyplývá že systém podporuje jen klíče s velikostí 4 bajty a více. Dále pak nejsou implementovány některé datové typy, pro rozsahové dotazy na R-strom. Jedná se o FLOAT (*cFloat)*, DOUBLE(*cDouble*) a SMALLINT (*cShort*).

Další problém na straně systému nastal u rozsahových dotazů, v některých případech návratová hodnota rozsahového dotazu neobsahovala všechny hodnoty, co by měla. To způsobuje problémy u vzoru 4, když při vytváření nového indexu na existující shlukovanou tabulku dotazujeme index primárního klíče. Počet prvků v indexu je správný, ale data nejsou vráceny všechny, tudíž při indexaci nastane chyba. Tato chyba nastává, když shlukovaná tabulka obsahuje více jak 250 000 záznamů.

10 Závěr

Cílem mé práce bylo vytvořit aplikaci, která vytvoří jednotlivé datové struktury pro uživatelský příkaz jazyka JDD a naplní ji daty pro testování. To se mi povedlo ne úplně dokonale, některé části mají poměrně vysokou režii a překladač je citlivý na překlepy ve vstupním příkazu, nicméně zadání jsem splnil a všechny vzory je má aplikace schopna vytvořit.

Z výsledků je patrné, že R-strom má lepší schopnost vkládání nových dat do databáze, při větším množství již existujících záznamů. Časové výkyvy prvního a druhého vkládáni jsou minimální. Nicméně u menšího množství záznamů je efektivnější B-strom. Vzor 1 je výhodný používat pro menší databáze s větším množstvím indexů na tabulky, díky rychlému indexování existujících záznamů. Rychlé indexování má i vzor 3 typu B-strom, ale následné vkládání je nejpomalejší z testovaných, kvůli zvýšené režii u vkládání záznamů do shlukovaných tabulek. Vzor 3 poskytuje výhody rychlejšího dotazování, které však v mé práci moc netestuji.

V této práci jsem narazil na velké množství zádrhelů, hlavně při využívání funkcí systému RadegastDB, nebo chybějící implementaci některých jeho částí, které jsem buď musel dopisovat, pokud se jednalo o triviální záležitosti, nebo jsem tyto problémy musel řešit s vedoucím práce.

Z mého pohledu byla moje práce velmi časově náročná. Tato práce pro mě byla velkou životní zkušeností. Obohatila mě o mnoho, hlavně o znalost principu fungování databázových systémů a celkově databází, proto jsem si toto téma vybral.

Literatura

[1] Jiří Dvorský. Algoritmy 1, VŠB - Technická univerzita Ostrava, 2007. [cit. 2017-06-27]

[2] HOLUŠA, David. Porovnání vlastností různých implementací uzlů datových struktur. VŠB - Technická univerzita Ostrava, 2015. [cit. 2017-06-27]

[3] C++ [online]. 2016 [cit. 2017-06-28]. Dostupné z: http://www.stroustrup.com/C++.html

[4] Serialization. Boost C++ Libraries [online]. Copyright © Copyright [cit. 01.06.2017]. Dostupné z: http://www.boost.org/doc/libs/1_64_0/libs/serialization/doc/index.html

[5] Embedded databáze - Úvod. ROOT.cz [online]. 2004, , 1 [cit. 2017-06-15]. Dostupné z: https://www.root.cz/clanky/embedded-databaze-uvod/

[6] Types And Classification Of Database Management System. What Is Database Management System (DBMS)? [online]. Copyright © Sumit Thakur [cit. 20.04.2017]. Dostupné z:http://whatisdbms.com/types-and-classification-of-database-management-system/

[7] Embedded database. Techopedia [online]. [cit. 2017-06-28]. Dostupné z: https://www.techopedia.com/definition/30660/embedded-database

[8] KRÁTKÝ, Michal a Radim BAČA. Databázové systémy [online]. VŠB – Technická univerzita Ostrava, 2009 [cit. 2017-06-15]. Dostupné z: http://dbedu.cs.vsb.cz/SubPages/OpenFile.aspx?file=book/dbcb.pdf

[9] Oracle Database SQL Reference [online]. 2003 [cit. 2017-06-28]. Dostupné z: https://docs.oracle.com/cd/B12037_01/server.101/b10759.pdf

[10] Data-definition-language-ddl. Study [online]. [cit. 2017-06-28]. Dostupné z: http://study.com/academy/lesson/data-definition-language-ddl-definition-example.html

[11] Query language. Techopedia [online]. [cit. 2017-06-28]. Dostupné z: https://www.techopedia.com/definition/3948/query-language

[12] Data-manipulation-language. Techopedia [online]. [cit. 2017-06-28]. Dostupné z: https://www.techopedia.com/definition/1179/data-manipulation-language-dml

[13] What is a System Catalog? - Definition from Techopedia. Techopedia - Where IT and Business Meet [online]. Copyright © 2017 Techopedia Inc.. [cit. 20.04.2017]. Dostupné z:https://www.techopedia.com/definition/22442/system-catalog

[14] E. Codd. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377–387, 1970. [cit. 2017-06-27]

[15] Distribuovaný databázový systém. Pdf.truni [online].[cit. 2017-06-27]. Dostupné z:http://pdf.truni.sk/e-ucebnice/databazove-systemy2/data/839db3f0-6971-43d5-bbbd-c697fcc4cfd6.html?ownapi=1

[16] The Codasyl Approach to Data Base Management. T. William Olle. Wiley, 1978. ISBN 0-471-99579-7. [cit. 2017-06-27]

[17] Paul E. Black: (a,b)-strom v encyklopedii algoritmů a datových struktur, U.S. National Institute of Standards and Technology, 6. 10. 2004, [cit. 2017-06-27]

[18] ACID. Postgres [online]. [cit. 2017-06-28]. Dostupné z: http://postgres.cz/wiki/Slovn%C3%ADk#ACID

[19] Databázové systémy. Homen.vsb [online]. [cit. 2017-06-27]. Dostupné z: http://homen.vsb.cz/~s1i95/isvdas/is/is_db_sys.htm

[20] SMOLKA, Michal. Webové rozhraní pro testování frameworku RadegastDB [online]. VŠB – Technická univerzita Ostrava Fakulta elektrotechniky a informatiky, 2016 [cit. 2017-06-28]. Dostupné z: https://dspace.vsb.cz/bitstream/handle/10084/116098/SMO0080_FEI_B2647_2612R025_2016.pdf?sequence=1&isAllowed=y. Bakalářská práce.

[21] Bača, R., Chovanec, P., Krátký, M., Lukáš, P. QuickDB - Yet another Database Management System? (2014) CEUR Workshop Proceedings, 1139, pp. 91-99. [cit. 2017-06-27]

[22] Lukáš, P., Bača, R., Krátký, M. QuickXDB: A prototype of a native XML DBMS (2013) CEUR Workshop Proceedings, 971, pp. 36-47. [cit. 2017-06-27]

[23] Database 12c | Oracle. [online]. Copyright © Oracle [cit. 15.06.2017]. Dostupné z: https://www.oracle.com/database/index.html

[24] MySQL. MySQL [online]. Copyright © 2017, Oracle Corporation and [cit. 15.06.2017]. Dostupné z: https://www.mysql.com/

[25] SQL Server 2016 | Microsoft. Microsoft Corporation [online]. Copyright © Microsoft 2017 [cit. 15.06.2017]. Dostupné z: https://www.microsoft.com/cs-cz/sql-server/sql-server-2016

[26] IBM DB2 – Database software – IBM Analytics . [online]. Dostupné z: https://www.ibm.com/analytics/us/en/technology/db2/

[27] Databázový software a aplikace | Microsoft Access. Object moved [online]. Copyright © Microsoft 2017 [cit. 15.06.2017]. Dostupné z: https://products.office.com/cs-cz/access

[28] Oracle Berkeley DB 12c [online]. [cit. 2017-06-18]. Dostupné z: http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html

[29] Firebird. Firebirdsql [online]. [cit. 2017-06-18]. Dostupné z: https://www.firebirdsql.org/pdfmanual/html/ufb-cs-embedded.html

[30] SQLite Release 3.19.3. Sqlite [online]. [cit. 2017-06-18]. Dostupné z: https://www.sqlite.org/releaselog/3_19_3.html

[31] CSQL [online]. [cit. 2017-06-25]. Dostupné z: http://csql.sourceforge.net/

[32] EXtremeDB. Mcobject [online]. [cit. 2017-06-25]. Dostupné z: http://www.mcobject.com/extremedbfamily.shtml

[33] Hsqldb [online]. 2017 [cit. 2017-06-25]. Dostupné z: http://hsqldb.org/

[34] Informix Dynamic Server. Ibm [online]. [cit. 2017-06-25]. Dostupné z: https://www.ibm.com/developerworks/data/roadmaps/roadmap_ids_10.html

[35] Infinitydb. Boilerbay [online]. 2017 [cit. 2017-06-25]. Dostupné z: https://boilerbay.com/infinitydb/

[36] InnoDB. Dev.mysql [online]. [cit. 2017-06-25]. Dostupné z: https://dev.mysql.com/doc/refman/5.5/en/innodb-storage-engine.html

[37] Interbase. Embarcadero [online]. [cit. 2017-06-25]. Dostupné z: https://www.embarcadero.com/products/interbase

[38] Raima [online]. 2017 [cit. 2017-06-25]. Dostupné z: http://raima.com/

[39] SQL Server Compact. Microsoft [online]. [cit. 2017-06-25]. Dostupné z: https://www.microsoft.com/cs-cz/download/details.aspx?id=17876

[40] Xquery [online]. 2014 [cit. 2017-06-28]. Dostupné z: https://www.w3.org/TR/xquery-30/

[41] Linq. Docs.microsoft [online]. 2017 [cit. 2017-06-28]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

[42] Datalog. Docs.racket-lang [online]. [cit. 2017-06-28]. Dostupné z: http://docs.racket-lang.org/datalog/

[43] OQL. Ibm [online]. [cit. 2017-06-28]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SSSHRK_4.2.0/itnm/ip/wip/ref/concept/pen_ref_oql.html

[44] QUEL query languages. Scribd [online]. [cit. 2017-06-28]. Dostupné z: https://www.scribd.com/document/53314850/QUEL-query-languages

[45] Sql:2016. Modern-sql [online]. [cit. 2017-06-28]. Dostupné z: https://modern-sql.com/blog/2017-06/whats-new-in-sql-2016

[46] Apache [online]. 2017 [cit. 2017-06-27]. Dostupné z: https://httpd.apache.org/

[47] Openldap [online]. 2017 [cit. 2017-06-27]. Dostupné z: https://www.openldap.org/

[48] Openoffice [online]. openoffice, 2017 [cit. 2017-06-27]. Dostupné z: https://www.openoffice.cz/

A Příloha na CD

Tabulka 8: Obsah CD

\DDLProcesor	balíček bakalářské práce
\DDLProcesor\DDLProcesor	řešení mé aplikace
\DDLProcesor\Framework	zdrojové kódy frameworku RadegastDB
\DDLProcesor\Boost.zip	archív obsahující zkompilovanou knihovnu boost
\DDLProcesor\DDLProcesor\testy.txt	sada testů pro vytváření datových struktur

Před prvním spuštěním aplikace, se musí vyextrahovat knihovna boost z archívu boost.zip aby struktura projektu po extrakci vypadala takto:

Tabulka 9: Struktura projektu po extrakci knihovny boost

\DDLProcesor	balíček bakalářské práce
\DDLProcesor\DDLProcesor	řešení mé aplikace
\DDLProcesor\Framework	zdrojové kódy frameworku RadegastDB
\DDLProcesor\Boost.zip	archív obsahující zkompilovanou knihovnu boost
\DDLProcesor\lib	zkompilované .lib soubory
\DDLProcesor\boost	zdrojové soubory boost knihovny
\DDLProcesor\DDLProcesor\testy.txt	sada testů pro vytváření datových struktur